Apple Assembly Line

$1.50

Volume 2 -- Issue 9 June, 1982

In This Issue...

Advertising in AAL

Due to the increased costs of printing more than 1600 copies per month, and with the desire to limit the percentage of advertising pages to less than 30% each month, I have decided to raise the page rate again.

For the July 1982 issue the price will be $50 for a full page, $30 for a half page. So-called "classified" ads, of up to forty words, will be $5.

About "The Other Epson Reference Manual"

I have received a number of complaints from readers that Cut the Bull Software Co. doesn't cut the mustard. Their $5 checks have been cashed, but after many weeks they still have not received the booklet they ordered. I have no phone number for either the company or the owner, and apparently neither does Information. I know the booklet exists, because I have one. I believe the company plans to fill all the orders, but procrastination has taken over. (They sent me a copy of another booklet, which I don't plan to review.) Until they see fit to fill their back-orders, and to publish a phone number, I don't recommend ordering their booklet.

Implementing New Opcodes Using 'BRK'.....Bob Sander-Cederlof

If you have the Autostart ROM, you can control what happens when a BRK instruction is executed. If you do nothing, a BRK will cause entry into the Apple Monitor, and the register contents will be displayed. But (if you have the Autostart Monitor) by a small amount of programming you can make the BRK do marvelous things.

Like simulate neat instructions from the 6809, which are not in the 6502, for example. I am thinking particularly of the LEAX instruction, which loads the effective address into a 16-bit register; of BSR, which enters a subroutine like JSR, but with a relative address; and of BRA, which is a relatively addressed JMP. With these three instructions you can write position-independent programs (programs that execute properly without any modification regardless of where they are loaded in memory).

I am thinking of these because of an article by A. Sato in "Lab Letters" (a publication of ESD Laboratories in Tokyo, JAPAN) Volume 6 No. 1, pages 91-93. It is all written in Japanese (see example below), but I think I deciphered what he is saying.

When a BRK instruction is executed, the program is interrupted as though a Non-Maskable Interrupt (NMI) occurred. The B bit in the status register is set, so the Apple can tell that the interrupt was caused by BRK rather than some external event. After making this determination, the Autostart Monitor performs a "JMP ($3F0)" instruction. This means that you can get control by placing the address of your own program into $3F0 and $3F1. The monitor initialization process puts the address $FA59 there.

By the time the monitor branches to the BRK processor (its own or yours) all the registers have been saved. The address of the BRK instruction plus 2 (PC) has been saved at $3A and $3B; the registers A, X, Y, P (status), and S (stack pointer) have been saved in $45 through $49, respectively.

BRK Interceptor/Interpreter

In the program below, lines 1180-1230 will set up the BRK-vector at $3F0 and $3F1 to point to your own BRK processor. Lines 1250-1320 back up the PC value by one, to point at the byte immediately following the BRK instruction. At this point I can decide what to do about the BRK.

Since I want to simulate the operation of LEAX, BSR, and BRA, I will use the BRK instruction to introduce a pseudo instruction of three bytes. I decided to copy A. Sato on this. LEAX is a BRK instruction followed by LDX from an absolute address. This is $AE in hexadecimal, followed by a 16-bit value representing a relative address. BSR is BRK followed by a JSR instruction ($20) and a relative address; BRA is BRK followed by a JMP instruction ($4C) and a relative address.

Looking back at the program, lines 1310 and 1320 store the address of the secondary opcode byte into PNTR and PNTR+1. These two bytes are inside an instruction at line 1760. I didn't want to use any page-zero space, so I had to resort to this kind of self-modifying code. While we are here, lines 1750-1780 pick up the byte whose address is in PNTR. Lines 1710-1740 increment PNTR. If we call GET.THIS.BYTE, it just picks up the byte currently pointed at. If we call GET.NEXT.BYTE, it increments the pointer and gets the next byte.

Lines 1330-1370 pick up the three bytes which follow the BRK. The opcode byte is saved in the Y-register. Lines 1380-1450 compute the effective address, by adding the actual address of the instruction to the relative address inside the instruction.

Lines 1470-1540 classify the opcode; if it is one of the three we have implemented, it branches to the appropriate code. If not, it jumps back into the monitor and processes the BRK in the normal monitor way.

Opcode Implementation

Lines 1560-1780 implement the three opcodes BSR, BRA, and LEAX. BRA (Branch Always) is the easiest one. We have already computed the effective address and stored it in the address field of the JMP instruction at line 1620. All BRA does is restore the registers (line 1610), and JMP to the effective address.

BSR (Branch to Subroutine) is only slightly harder. We first have to push the return address on the stack, and then do a BRA. Lines 1560-1590 do the pushing.

LEA (Load Effective Address) is the hardest. Lines 1650-1690 do the work. First GET.NEXT.BYTE moves the address in PNTR,PNTR+1 to point at the first byte of the next instruction. That is so we can continue exectution. Then MON.RESTORE gets back the original contents of all the registers. THEN LDY and LDX pick up the effective address in the Y- and X-registers. The high byte of the effective address is in the X-register, and the Z- and N-bits in the status register reflect the value of this byte. If you wish, you could modify this to not change the status by inserting a PHP opcode after line 1660, and PLP after line 1680; then the status register would remain unchanged by the entire LEA process. Or you could reverse lines 1670 and 1680, so that the status reflected the low-order byte of the effective address.

Demonstration Using the New Opcodes

Lines 1800 and beyond are a demonstration of the use of the new opcodes. First I defined some macros for the new opcodes. I didn't have to do this, but it is convenient if you have a macro assembler. If you don't, you can use the BRK instruction

on one line, followed by a LDX, JSR, or JMP instruction with a relative address on the next line.

My macros are defined in a nested fashion. The BRK macro generates two lines: BRK on the first line, and a second line consisting of the specified opcode and operand. The LEA, BSR, and BRA macros call BRK to generate LDX, JSR, and JMP instructions after the BRK. The operand field is a relative address, computed within the BRK macro.

The demonstration program will run anywhere in memory, as long as the BRK interpreter has been loaded and initialized. You can test this by moving $871-89F to other places and running it. What it does is print out the message in line 2090.

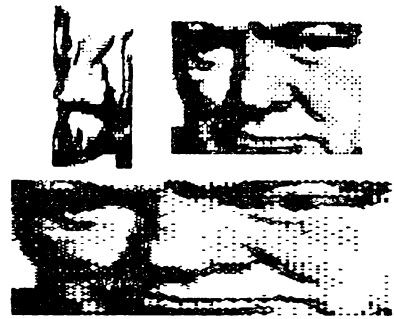
```
                    1010 *--------------------------------
                    1020 *      IMPLEMENTING BSR, BRA, AND LEA OPCODES
                    1030 *      USING THE 'BRK' VECTOR WITH THE
                    1040 *      AUTOSTART ROM
                    1050 *
                    1060 *      ADAPTED FROM AN ARTICLE IN "LAB LETTERS"
                    1070 *           BY A. SATO
                    1080 *--------------------------------
003A-               1090 MON.PC     .EQ $3A,3B
0046-               1100 MON.XREG   .EQ $46
0047-               1110 MON.YREG   .EQ $47
                    1120 *--------------------------------
03F0-               1130 BRK.VECTOR .EQ $3F0,3F1
                    1140 *--------------------------------
FA59-               1150 MON.BRK    .EQ $FA59
FF3F-               1160 MON.RESTORE .EQ $FF3F
                    1170 *--------------------------------
                    1180 SETUP
0800- A9 0B         1190        LDA #BREAK.INTERPRETER
0802- 8D F0 03      1200        STA BRK.VECTOR
0805- A9 08         1210        LDA /BREAK.INTERPRETER
0807- 8D F1 03      1220        STA BRK.VECTOR+1
080A- 60            1230        RTS
                    1240 *--------------------------------
                    1250 BREAK.INTERPRETER
080B- A4 3B         1260        LDY MON.PC+1     PICK UP ADDRESS OF THIRD BYTE
080D- A6 3A         1270        LDX MON.PC
080F- D0 01         1280        BNE .1       BACK UP TO SECOND BYTE
0811- 88            1290        DEY
0812- CA            1300 .1     DEX
0813- 8E 6B 08      1310        STX PNTR     MODIFY ADDRESS IN GET.THIS.BYTE SUBRO
0816- 8C 6C 08      1320        STY PNTR+1
0819- 20 6A 08      1330        JSR GET.THIS.BYTE
081C- A8            1340        TAY          OPCODE BYTE
081D- 20 62 08      1350        JSR GET.NEXT.BYTE
0820- 48            1360        PHA          ADDR-LOW BYTE
0821- 20 62 08      1370        JSR GET.NEXT.BYTE
0824- AA            1380        TAX
0825- 68            1390        PLA
0826- 38            1400        SEC          ADDR-HIGH BYTE
0827- 6D 6B 08      1410        ADC PNTR     COMPUTE EFFECTIVE ADDRESS
082A- 8D 51 08      1420        STA EFF.ADDR
082D- 8A            1430        TXA
082E- 6D 6C 08      1440        ADC PNTR+1
0831- 8D 52 08      1450        STA EFF.ADDR+1
                    1460 *--------------------------------
0834- C0 20         1470        CPY #$20     CLASSIFY OPCODE
0836- F0 0D         1480        BEQ BSR
0838- C0 4C         1490        CPY #$4C
083A- F0 11         1500        BEQ BRA
083C- C0 AE         1510        CPY #$AE
083E- F0 13         1520        BEQ LEA
0840- 84 47         1530        STY MON.YREG
0842- 4C 59 FA      1540        JMP MON.BRK
                    1550 *--------------------------------
```

```
0845- AD 6C 08 1560 BSR     LDA PNTR+1   PUSH RETURN ADDRESS ON STACK
0848- 48      1570          PHA
0849- AD 6B 08 1580          LDA PNTR
084C- 48      1590          PHA          AND DO BRA
              1600 *-------------------------------
084D- 20 3F FF 1610 BRA     JSR MON.RESTORE
0850- 4C 00 00 1620          JMP *-*
0851-         1630 EFF.ADDR   .EQ *-2
              1640 *-------------------------------
0853- 20 62 08 1650 LEA     JSR GET.NEXT.BYTE  POINT AT NEXT INSTRUCTION
0856- 20 3F FF 1660          JSR MON.RESTORE   RESTORE A-REG AND STATUS
0859- AC 51 08 1670          LDY EFF.ADDR      ADDR-LO IN Y
085C- AE 52 08 1680          LDX EFF.ADDR+1    ADDR-HI IN X
085F- 6C 6B 08 1690          JMP (PNTR)
              1700 *-------------------------------
              1710 GET.NEXT.BYTE
0862- EE 6B 08 1720          INC PNTR
0865- D0 03    1730          BNE GET.THIS.BYTE
0867- EE 6C 08 1740          INC PNTR+1
              1750 GET.THIS.BYTE
086A- AD FF FF 1760          LDA $FFFF    (FILLED IN)
086B-         1770 PNTR     .EQ *-2
086D- 60      1780          RTS
              1790 *-------------------------------
0000-         1800 MSG      .EQ 0,1
086E- 4C ED FD 1810 JMP.COUT JMP $FDED
              1820          .MA LEA
              1830          >BRK LDX,]1
              1840          .EM
              1850          .MA BSR
              1860          >BRK JSR,]1
              1870          .EM
              1880          .MA BRA
              1890          >BRK JMP,]1
              1900          .EM
              1910          .MA BRK
              1920          BRK
              1930          ]1 ]2-:1
              1940 :1
              1950          .EM
0871-         1960 TEST     >LEA MESSAGE
0871-         0000>          >BRK LDX,MESSAGE
0871- 00      0000>>          BRK
0872- AE 19 00 0000>>          LDX MESSAGE-:1
              0000>> :1
0875- 86 01    1970          STX MSG+1
0877- 84 00    1980          STY MSG
0879- A0 00    1990          LDY #0
087B- B1 00    2000 .1       LDA (MSG),Y
087D- 48       2010          PHA
087E- 09 80    2020          ORA #$80
0880-          2030          >BSR JMP.COUT
0880-          0000>          >BRK JSR,JMP.COUT
0880- 00       0000>>          BRK
0881- 20 EA FF 0000>>          JSR JMP.COUT-:1
               0000>> :1
0884- C8       2040          INY
0885- 68       2050          PLA
0886- 10 F3    2060          BPL .1
0888- A9 8D    2070          LDA #$8D     CARRIAGE RETURN
088A-          2080          >BRA JMP.COUT
088A-          0000>          >BRK JMP,JMP.COUT
088A- 00       0000>>          BRK
088B- 4C E0 FF 0000>>          JMP JMP.COUT-:1
               0000>> :1
088E- 54 48 49
0891- 53 20 49
0894- 53 20 4D
0897- 59 20 4D
089A- 45 53 53
089D- 41 47 C5 2090 MESSAGE .AT /THIS IS MY MESSAGE/
```

Autostart ROMでは、BRKベクトルがRAMを経由するため、ユーザとしては、使い勝手が広まって良いと思います。筆者も、かねてよりおもしろい使い方はないものかと考えてましたが、最近我家のAppleがM6809を頭脳にして走る様になり

EPSON HIEROGRAPHIC TRANSPORT

PRINTER GRAPHICS SOFTWARE



TRANSPORT FEATURES: WINDOW CONTROL ON HI-RES SCREEN (AS SHOWN ABOVE) ALLOWING ANY RECTANGULAR AREA OF THE SCREEN TO BE SELECTED FOR PRINT; VARIABLE SCALE IN BOTH HEIGHT AND WIDTH OF THE PICTURE ALLOWING STRETCH AND MAGNIFICATION; NORMAL OR ROTATED PICTURE; POSITIVE OR NEGATIVE PRINT; PICTURE FILES LOADED WITH A SINGLE KEY PRESS; MODULE INCLUDED THAT MAY BE CALLED FROM BASIC; WRITTEN IN MACHINE LANGUAGE.

FOR 48K APPLE II/APPLE II PLUS WITH EPSON MX-70/80/100 PRINTER (MX-80 MUST HAVE GRAFTRAX 80 GRAPHICS OPTION)

APPLE II / APPLE II PLUS ARE TRADEMARKS OF APPLE COMPUTER, INC.
EPSON MX-70/80/100 AND GRAFTRAX 80 GRAPHICS ARE TRADEMARKS OF EPSON AMERICA, INC.

THE CLONE KIT

NOW $34.5

CLONE KIT IS A MULTI-PURPOSE BIT COPIER FOR THE APPLE II COMPUTER, GIVING YOU THE MEANS TO PROTECT YOUR INVESTMENT IN SOFTWARE BY BACKING UP MOST OF YOUR "PROTECTED" DISKETTES.

WITH CLONE KIT YOU CAN:

- BACKUP DISKETTES USING SYNCHRONIZED TRACKS AS PROTECTION
- BACKUP DISKETTES USING A MODIFIED DOS
- COPY ONLY SELECTED TRACKS (USEFUL TO RESTORE A "BLOWN" DOS)
- MAKE A SINGLE DRIVE BACKUP
- DUPLICATE DISKS USING SPECIAL TIMING FOR PROTECTION
- PROVIDES EXTENDED INFORMATION FOR VIEWING THE DISKETTE STRUCTURE
- COPY PASCAL AND CP/M DISKETTES QUICKLY AND EASILY

WORKS WITH BOTH 13 AND 16 SECTOR DISKETTES

WRITE OR CALL (214) 259-6482 OR (214) 259-5196

HIEROGRAPHIC TRANSPORT ... $39.95
THE CLONE KIT ..... ~~$49.95~~ $34.95

GSR ASSOCIATES - P.O. BOX 401462 - GARLAND, TEXAS 75040

DEALER INQUIRIES INVITED

Checks, M.O.

A New Hi-Res Function for Applesoft..... Mike Laumer

Most people use the language card as nothing more than a ROM simulator for the other version of BASIC that is not on the motherboard. But it can do much more since the memory is actually RAM. Indeed Bob S-C's Macro Assembler has a version which runs in a Language Card. The FLASH! Integer BASIC compiler which I wrote uses the language card in place of a disk file providing higher speed compilations for those people who have a language card.

One nice aspect of having the language card is the ability to move Apple software from ROM to RAM in the card and make changes to add a new capability. Some people have done this already with the Apple monitor to add an extra feature or two at the expense of another (who needs the tape I/O routines).

The program assciated with this article will allow you to patch a RAM card version of Applesoft to modify the 'HPLOT' command to function as an 'HXPLOT' command. What is 'HXPLOT' you say. Remember the DRAW and XDRAW commands in Applesoft. The 'DRAW' command will place a shape on the screen; 'XDRAW' does the same thing, but 'XDRAW' has the unique ability to redraw the shape and erase it from the screen leaving whatever was on the screen initially still intact. The 'HXPLOT' function in the listing functions the same way for the 'HPLOT' command as 'XDRAW' does for the 'DRAW' command.

I have been developing a Hi-Res graphics editor as my next product. During the development cycle I was working with a line draw game paddle routine. You move a cursor to a position and anchor one end of the line to a point. Then you can move to another position and while you move a line stretches out from the point like a rubber band to the current cursor position. This gives you a preview of what the line looks like before you plot the line. The 'HXPLOT' function does have one sleight problem: it plots independent of the current color.

What the function actually does as it draws a line is to invert each dot of the line path instead of plotting a color. When the same line is drawn with the same coordinates the bits on the line path are inverted again back to their original value, restoring the screen to what it was before you started HXPLOTting.

You may be wondering why not just use the 'HPLOT' as it is to do this. You could just draw the line once with a color of 3 then change the color to 0 and erase the line with another 'HPLOT'. This only works if you have a black screen with no other images on it. If their are other images on the screen then when you erase the line you will draw a black line through those other images causing them to change. Only a function like 'XDRAW' or the 'HXPLOT' will be non-destructive of the background data on the screen.

How It Works

The 'HPLOT' command in Applesoft is actually two commands in one.

```
HPLOT x, y              plots 1 point
HPLOT x1,y1 TO x2,y2    plots a line
```

Each of the routines have one common place where they plot a bit onto the hi-res screen. The point plotting routine is at $F457 in the ROM and the line routine is at $F53A in the ROM. By putting Applesoft into the RAM card we can patch into these routines and modify their operation.

The two areas that are patched are at $F457 and $F58D. After you run the patch program you should see the Applesoft prompt character and there will be no program in memory. So type in the small demo program listed here and run it.

```
10  HGR2 : POKE 769,1
15  POKE 28,127: CALL 62454
20  FOR I = 0 TO 279 STEP 10: FOR J = 0 TO 192 STEP 10
25  HPLOT 140,96 TO I,J
30  FOR Z = 1 TO 1: NEXT Z
40  HPLOT 140,96 TO I,J
50  NEXT J: NEXT I
100  GOTO 10
```

If you have an Integer BASIC motherboard you should boot up your system master disk and have Applesoft loaded into your RAM card before using the routine.

```
                    1000 *--------------------------------
                    1010 * THIS ROUTINE ADDS AN XPLOT CAPABILITY
                    1020 * TO APPLESOFT.  THE FLAG AT $301 (769)
                    1030 * CONTROLS WHETHER HPLOT OR XPLOT IS
                    1040 * FUNCTIONING.
                    1050 *
                    1060 *       POKE 769,0   ENABLES HPLOT
                    1070 *       POKE 769,1   ENABLES XPLOT
                    1080 *--------------------------------
                    1090         .OR $300
                    1100         .TF B.HXPLOT
0300- A9 00         1110 NEW.HLIN LDA #0      TEST 'XPLOT' FLAG
0302- D0 09         1120         BNE .2       YES 'XPLOT' MODE
0304- B1 26         1130         LDA ($26),Y  PLOT NORMAL LINE
0306- 45 1C         1140         EOR $1C
0308- 25 30         1150         AND $30
030A- 4C 93 F5      1160 .1      JMP $F593    BACK INTO APPLESOFT LINE ROUTINE
030D- A9 7F         1170 .2      LDA #$7F     MASK COLOR SHIFT BIT
030F- 25 30         1180         AND $30      OFF OF BIT MASK
0311- 31 26         1190         AND ($26),Y  TEST SCREEN BIT
0313- D0 F5         1200         BNE .1       BIT IS SET!... SO CLEAR IT
0315- A9 7F         1210         LDA #$7F     BIT IS CLEAR!...SO SET IT
0317- 25 30         1220         AND $30      BIT MASK WITHOUT COLOR SHIFT BIT
0319- 10 EF         1230         BPL .1       BRANCH ALWAYS
                    1240 *--------------------------------
031B- 20 11 F4      1250 NEW.PLOT JSR $F411   CALL HPOSN ROUTINE
031E- AD 01 03      1260         LDA $301     TEST 'XPLOT' FLAG
0321- D0 03         1270         BNE .1       YES 'XPLOT' MODE
0323- 4C 5A F4      1280         JMP $F45A    PLOT NORMAL
0326- A9 7F         1290 .1      LDA #$7F     XPLOT
0328- 25 30         1300         AND $30      MASK COLOR SHIFT BIT OFF
032A- 31 26         1310         AND ($26),Y  TEST SCREEN BIT
032C- D0 04         1320         BNE .2       SCREEN BIT IS SET
032E- A9 7F         1330         LDA #$7F     ...CLEAR SO PREPARE TO
0330- 25 30         1340         AND $30      SET SCREEN BIT
0332- 4C 60 F4      1350 .2      JMP $F460    BACK INTO APPLESOFT XPLOT ROUTINE
                    1360 *--------------------------------
                    1370 *
                    1380 * TO USE THE ABOVE FUNCTION YOU MUST HAVE A RAM CARD.
                    1390 * APPLESOFT MUST BE IN THE RAM CARD.
                    1400 * THEN YOU MUST DO THE FOLLOWING:
                    1410 *
                    1420 * 0. BLOAD B.XPLOT.FOR.FP     LOAD THE XPLOT ROUTINE
                    1430 * 1. CALL-151  TO ENTER THE MONITOR
                    1440 * 2. C081 C081   TO WRITE ENABLE THE CARD
                    1450 * 3. GO TO STEP 5 IF YOU HAVE AN INTEGER BASIC MOTHER
                    1460 * 4. D000<D000.FFFFM   PUT APPLESOFT INTO RAM CARD
                    1470 * 5. F58D:4C 00 03   PATCH FOR LINE ROUTINE
                    1480 * 6. F457:4C 1B 03   PATCH FOR POINT PLOT ROUTINE
                    1490 * 7. C080            WRITE PROTECT THE RAM CARD
                    1500 * 8. 3D3G            START APPLESOFT UP
                    1510 *--------------------------------
                    1520 * FOR LAZY SOULS HERE IS AN AUTOMATIC PATCH ROUTINE.
                    1530 *--------------------------------
FDED-               1540 MON.COUT .EQ $FDED   MONITOR CHARACTER OUT ROUTINE
                    1550         .OR $4000
                    1560         .TF B.PATCH.XPLOT
4000- A0 00         1570 START   LDY #0
4002- B9 0D 40      1580 .1      LDA MESG,Y
4005- F0 1E         1590         BEQ L.100
4007- 20 ED FD      1600         JSR MON.COUT PRINT MESSAGE
400A- C8            1610         INY
400B- D0 F5         1620         BNE .1       BRANCH ALWAYS
400D- 8D 84         1630 MESG    .HS 8D84
400F- C2 CC CF
4012- C1 C4 A0
4015- C2 AE D8
4018- D0 CC CF
401B- D4 AE C6
401E- CF D2 AE
4021- C6 D0         1640         .AS -/BLOAD B.XPLOT.FOR.FP/
4023- 8D 00         1650         .HS 8D00
```

```
4025- AD 81 C0 1660 L.100  LDA $C081     ROM READ
4028- AD 81 C0 1670        LDA $C081     RAM CARD WRITE
402B- AD 00 E0 1680        LDA $E000     CHECK MOTHERBOARD ROM
402E- C9 20    1690        CMP #$20      IS IT INTEGER BASIC
4030- F0 15    1700        BEQ L.200     YES SO MUST HAVE FP FROM SYSTEM MASTER
4032- A9 D0    1710        LDA #$D0      NO SO COPY FP FROM ROM TO RAM CARD
4034- 85 01    1720        STA $1
4036- A9 00    1730        LDA #0
4038- 85 00    1740        STA $0
403A- A0 00    1750 .1     LDY #0
403C- B1 00    1760 .2     LDA ($0),Y
403E- 91 00    1770        STA ($0),Y
4040- C8       1780        INY
4041- D0 F9    1790        BNE .2
4043- E6 01    1800        INC $1
4045- D0 F3    1810        BNE .1
4047- A9 4C    1820 L.200  LDA #$4C      SET PATCHES INTO RAM CARD APPLESOFT
4049- 8D 8D F5 1830        STA $F58D
404C- 8D 57 F4 1840        STA $F457
404F- A9 00    1850        LDA #NEW.HLIN
4051- 8D 8E F5 1860        STA $F58E
4054- A9 03    1870        LDA /NEW.HLIN
4056- 8D 8F F5 1880        STA $F58F
4059- A9 1B    1890        LDA #NEW.PLOT
405B- 8D 58 F4 1900        STA $F458
405E- A9 03    1910        LDA /NEW.PLOT
4060- 8D 59 F4 1920        STA $F459
4063- AD 80 C0 1930        LDA $C080
4066- 4C D3 03 1940        JMP $3D3      START UP RAM CARD APPLESOFT
```

Bubble Sort Demonstration Program.....Bob Sander-Cederlof

The following program implements one of the most inefficient methods of sorting a list of items ever invented. It is also a very specific implementation, not general at all. But it should be valuable to study if you are not already well-versed in sorting techniques. After execution, the bytes from $00 to $1F will be in ascending order.

```
                    1000 *--------------------------------
                    1010 *       BUBBLE-SORT DEMO
                    1020 *--------------------------------
0000-               1030 LIST    .EQ $00 THRU $0F
0010-               1040 N       .EQ 16
0010-               1050 FLAG    .EQ $10
                    1060 *--------------------------------
0800- A0 00         1070 BUBBLE LDY #0        INITIAL INDEX
0802- 84 10         1080        STY FLAG      INTERCHANGE FLAG
0804- B9 01 00      1090 .1     LDA LIST+1,Y COMPARE TWO ADJACENT ITEMS
0807- D9 00 00      1100        CMP LIST,Y
080A- B0 0F         1110        BCS .2        ALREADY IN CORRECT ORDER
080C- 48            1120        PHA           INTERCHANGE THEM
080D- B9 00 00      1130        LDA LIST,Y
0810- 99 01 00      1140        STA LIST+1,Y
0813- 68            1150        PLA
0814- 99 00 00      1160        STA LIST,Y
0817- A9 FF         1170        LDA #$FF      SET INTERCHANGE FLAG
0819- 85 10         1180        STA FLAG
081B- C8            1190 .2     INY           NEXT OVERLAPPING PAIR
081C- C0 0F         1200        CPY #N-1
081E- 90 E4         1210        BCC .1        STILL A PAIR LEFT
0820- A5 10         1220        LDA FLAG      WAS AN INTERCHANGE PERFORMED?
0822- D0 DC         1230        BNE BUBBLE    YES, MAKE ANOTHER PASS
0824- 60            1240        RTS           NO, ALL SORTED
```

DOS File Exchange: A Review.....Bill Morgan

I've just been playing with a new program called DOS File Exchange (DFX), and it is wonderful. Author Graeme Scott has provided a very useful tool for transferring any type of files through a modem, with full error-checking. You can even chat at the keyboards while the transfer is going on!

The DFX program must be running on both computers, and one of them must be using an original (primary) disk of the program. The program can be copied to produce a secondary disk; DFX will even send a copy of itself to a remote Apple, but the copy will be a secondary.

To transfer files, one user selects a "master" mode, so he will control both Apples. He then chooses whether he will send or receive; the program then transmits the sending Apple's disk catalog to the receiver. The master user selects the files wanted from the catalog and starts the transfer. Both users are then free to chat, supervise the transfer in one of three display modes, or even leave the room.

At almost any time, you can switch back and forth between Function and Chat modes. Function is used to select all control and menu choices; Chat sends all characters entered to the other Apple.

There are three display modes, called M(enu), U(tility), and G(raphic). Menu shows choices, including the disk catalog when files are being chosen. Utility displays the transmitted and received data streams, and allows more space for chatting. Graphic displays the data being transferred on the Hi-res screen, so if you are receiving a picture you can watch it take shape.

The only drawbacks I've found are that DFX will only operate with a Hayes Micromodem II in slot 2 and the disk in slot 6, drive 1.

DFX is available from Arrow Micro Software, 11 Kingsford, Kanata Ont., K2K 1T5 Canada.

Macro Hint.....Bob and Bill

For an easy semi-automatic SAVE, we use the following line in every program:

```
1000 *HHHHHHSAVE filename
```

The six H's are control H's (backspaces), entered by holding the CTRL key down and typing OHOHOHOHOHOH. (Control-O allows a following control character to be entered into a line.) To save the source file, just type LIST 1000, esc-I, and copy over the line. Make it a point to always have the SAVE in line 1000; it's much easier to remember.

Yes/No Subroutine.....Bob Sander-Cederlof

It happens all the time! I am continually needing to ask Yes/No questions in my programs. I do it now with the following subroutine, which has been somewhat stripped down for publication.

Assume you have just printed the question itself on the screen, preferably with " (Y/N)?" on the end. Then call my subroutine with "JSR YES.NO". The subroutine will clear the keyboard strobe, so that it is sure it is getting the answer to this question, and not just a stray character you accidentally typed. Then as soon as you hit any key, it will put it on the screen where the question ended and return to you.

At the point you should use BNE to branch where you want to if the user has typed something other than "Y" or "N". Once that is out of the way, use BCC or BCS to branch on whether it was "Y" or "N". The subroutine sets carry for "N" and clears carry for "Y".

In my actual programs, I have one more line between 1120 and 1130. It is JSR MESSAGE.PRINTER, which expects a message number in the Y-register. You can use it either way. You might also like to insert two more lines to call the message printer to print " (Y/N)? " for every question; that way the common string does not have to be repeatedly stored in memory with every question.

```
                        1000 *--------------------------------
                        1010 *       YES/NO SUBROUTINE
                        1020 *
                        1030 *       RETURN .NE. IF NEITHER "Y" NOR "N"
                        1040 *              .EQ. AND .CC. IF "Y"
                        1050 *              .EQ. AND .CS. IF "N"
                        1060 *--------------------------------
C010-                   1070 STROBE     .EQ $C010
FD0C-                   1080 MON.RDKEY  .EQ $FD0C
0024-                   1090 MON.CH     .EQ $24
0028-                   1100 MON.BASE   .EQ $28 AND $29
                        1110 *--------------------------------
                        1120 YES.NO
0800- 8D 10 C0          1130            STA STROBE
0803- 20 0C FD          1140            JSR MON.RDKEY
0806- A4 24             1150            LDY MON.CH
0808- C9 CE             1160            CMP #'N+$80
080A- F0 05             1170            BEQ .1
080C- C9 D9             1180            CMP #'Y+$80
080E- D0 03             1190            BNE .2
0810- 18                1200            CLC
0811- 91 28             1210 .1         STA (MON.BASE),Y
0813- 60                1220 .2         RTS
```

My Own Little Bell.....Bob Sander-Cederlof

The other day I was working on my Apple at home, and the kids were trying to sleep in the same room. The program I was working on needed to indicate erroneous input by a bell, and I had to test it. Suddenly I realized how loud and sharp the Apple bell is!

With all that motivation, I threw together this little routine which makes a soft and pleasant tone to use for my own little bell. It generates fifty repetitions of a triple-toggle pattern, with time intervals selected for their harmonious character.

Lines 1070, 1170, and 1180 establish a loop equivalent to the Applesoft code:

```
FOR X = 50 TO 1 STEP -1:        : NEXT
```

In assembly language it frequently occurs that backwards running loop counts are easier to use than forward ones, and this is just such a case.

Examine lines 1080-1160, and you will see a pattern repeated three times. In each case I load A with a value, call MON.DELAY, and toggle the speaker. The value passed to MON.DELAY is first 14, then 10, and then 6. MON.DELAY is a subroutine in the Apple Monitor ROM which delays an amount of time depending on what value you pass in the A-register, according to the following formula:

```
# cycles delay = 2.5*N*N + 13.5*N +13
```

This includes the six cycles of the JSR used to call the subroutine. Each cycle is...well, the Apple clock is roughly 1.023 MHz...so a cycle is about .9775 microseconds long. The counts of 14, 10, and 6 give intervals between toggles of 630.5, 204, and 195 (including the overhead instructions in SC.BELL).

You can play with the values, and try creating your own variations. You might try adding a fourth toggle per loop, changing the number of loops, changing the delay counts, and so on. Have fun!

```
                      1000 *--------------------------------
                      1010 *      MY OWN LITTLE BELL
                      1020 *--------------------------------
          FCA8-       1030 MON.DELAY  .EQ $FCA8
          C030-       1040 SPEAKER    .EQ $C030
                      1050 *--------------------------------
                      1060 SC.BELL
0800- A2 32           1070        LDX #50
0802- A9 0E           1080 .1     LDA #14
0804- 20 A8 FC        1090        JSR MON.DELAY
0807- AD 30 C0        1100        LDA SPEAKER
080A- A9 0A           1110        LDA #10
080C- 20 A8 FC        1120        JSR MON.DELAY
080F- AD 30 C0        1130        LDA SPEAKER
0812- A9 06           1140        LDA #6
0814- 20 A8 FC        1150        JSR MON.DELAY
0817- AD 30 C0        1160        LDA SPEAKER
081A- CA              1170        DEX
081B- D0 E5           1180        BNE .1
081D- 60              1190        RTS
```

Using the Shift-Key Mod.....Bob Sander-Cederlof

Have you heard of the "Shift-Key Mod"? By running a wire from the game connector to the right spot on the keyboard circuit, you can use software to tell whether or not the shift key is pressed. You can make your Apple keyboard almost normal!

Some word processors come with a convenient device which has a clip on one end of a wire, and a DIP socket-plug on the other. (I sell such a device for $15 without any software.) Apples with the piggy-back board below the keyboard can use the clip. If you don't have that kind of Apple, you need to solder a small wire to the bottom of either shift key, and clip onto that wire. Of course, you can run the wire all the way to the game connector and avoid the extra expense...I did it that way on my first Apple.

But what about software? All the mod does is bring the shift key into the game connector as PB2. You can read it with LDA $C063. If the value read is $00-7F, the shift key is being pressed; if $80-FF, the shift key is not pressed. You have to write a special keyboard input subroutine to convert letters to lower case ASCII codes if the shift key is not down.

Here is just such a subroutine! It is the one I use in my word processor (a product still being developed). Another routine sets up a cursor on the screen, and then calls READ.KEY.WITH.CASE to get the next keypress.

Lines 1140-1160 read the keyboard, and keep reading until you press a key other than the shift key. Once you press a key, the value at KEYBRD will be a code between $80 and $DF; the value is considered negative by the 6502, so execution continues at line 1170.

Lines 1170-1200 are an optional keyclick routine. In my word processor, a control-P turns the keyclicking on and off. I discovered that a very short "bell" sounds like a clicking keyboard, so that is what I use. The monitor bell subroutine toggles the speaker 192 times at about a 1000 Hertz rate to make a beep; I do it 10 times to make a click.

Lines 1210-1220 pick up the keypress code again and clear the keyboard strobe. This sets up the keyboard electronics so that you can read the next keypress next time around.

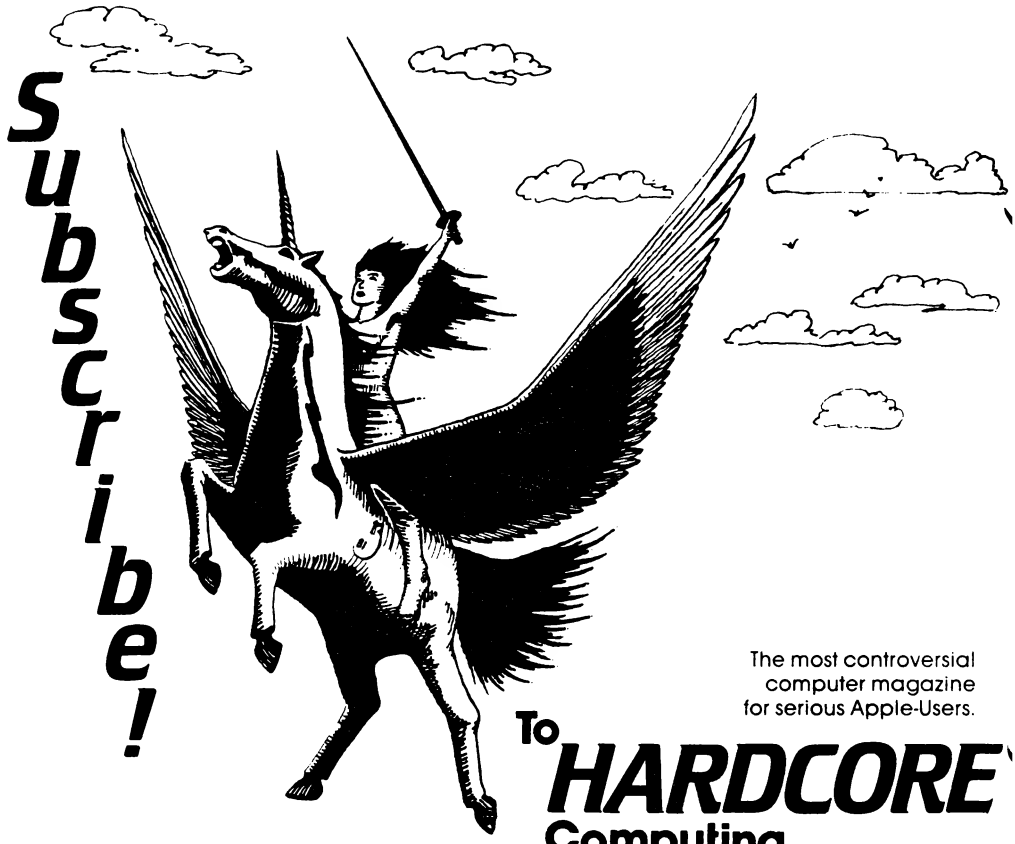
Lines 1230-1240 test the shift key. If it is down, the BPL will branch to the upper case section at line 1320. If the shift key is not down, lines 1270-1280 test whether the character is a letter. If so, line 1290 makes it into a lower-case code.

I am using the codes from $E0 through $FF for lower-case. This is standard ASCII, and is also compatible with the various lower-case display adapters available on the Apple. $E1 through $FA are the letters a-z; $E0 is a tick-mark; $FB-FF are special punctuation marks. If you don't have a lower-case display adapter, these codes display as punctuation and numbers.

Lines 1320-1420 handle characters typed with the shift key down. If the code is less than $C0, the keyboard input code is correct already. Above $C0, the code is correct unless you have typed M, N, or P. The Apple translates these shifted letters into @,], and ^, respectively. My logic translates them back into capital letters.

I use a special control sequence to enter the punctuation characters with codes above $C0, which is not shown here. You type control-O, which stands for "override", and then one of the letters klmnop or KLMNOP. The letter translates into the corresponding punctuation code. For example, control-O, shift-M is a right bracket (]); control-O, shift-P is an at-sign (@).

```
                        1000 *--------------------------------
                        1010 *       READ KEY WITH CASE CONTROL
                        1020 *--------------------------------
C000-                   1030 KEYBRD      .EQ $C000
C010-                   1040 KYSTRB      .EQ $C010
C030-                   1050 SPKR        .EQ $C030
C063-                   1060 SHIFT.KEY   .EQ $C063
                        1070 *--------------------------------
FBE4-                   1080 MON.BELL2   .EQ $FBE4
                        1090 *--------------------------------
0000-                   1100 KEY.CLICK.FLAG      .EQ $00
0001-                   1110 CASE.INPUT.FLAG     .EQ $01
0002-                   1120 CURRENT.CHAR        .EQ $02
                        1130 *--------------------------------
                        1140 READ.KEY.WITH.CASE
0800- AD 00 C0          1150         LDA KEYBRD   GET CHAR FROM KEYBOARD
0803- 10 FB             1160         BPL READ.KEY.WITH.CASE
0805- A5 00             1170         LDA KEY.CLICK.FLAG   CLICKING?
0807- F0 05             1180         BEQ .1       NO
0809- A0 0A             1190         LDY #10      YES, 10 HALF-CYCLES WILL
080B- 20 E4 FB          1200         JSR MON.BELL2    SOUND LIKE A CLICK
080E- AD 00 C0          1210 .1      LDA KEYBRD   CHAR AGAIN
0811- 8D 10 C0          1220         STA KYSTRB
0814- 2C 63 C0          1230         BIT SHIFT.KEY    SHIFT KEY DOWN?
0817- 10 0C             1240         BPL .2       YES
0819- 24 01             1250         BIT CASE.INPUT.FLAG
081B- 30 08             1260         BMI .2       IN SHIFT LOCK UPPER CASE
081D- C9 C0             1270         CMP #$C0     NO, LOWER CASE IF LETTER
081F- 90 18             1280         BCC .5       NOT A LETTER
0821- 09 20             1290         ORA #$20     LETTER, MAKE LOWER CASE
0823- D0 14             1300         BNE .5       ...ALWAYS
                        1310 *---SHIFT KEY PRESSED-------------
0825- C9 C0             1320 .2      CMP #$C0     SEE IF LETTER
0827- 90 10             1330         BCC .5       NOT A LETTER KEY
0829- F0 0C             1340         BEQ .4       SHIFT-P
082B- C9 DD             1350         CMP #$DD     SHIFT-M
082D- F0 04             1360         BEQ .3       YES
082F- C9 DE             1370         CMP #$DE     SHIFT-N
0831- D0 06             1380         BNE .5       NO
0833- 29 EF             1390 .3      AND #$EF     MAKE CAPITAL-M OR -N
0835- D0 02             1400         BNE .5       ...ALWAYS
                        1410 *--------------------------------
0837- A9 D0             1420 .4      LDA #$D0     MAKE CAPITAL-P
0839- 85 02             1430 .5      STA CURRENT.CHAR
083B- 60                1440         RTS
```



The most controversial computer magazine for serious Apple-Users.

To HARDCORE Computing

Isn't it time that you found out the true story behind the most controversial user-oriented computer magazine around today? Over 64 pages of facts, programming aids and program listings that includes columns on HOW TO: 1. Copy-protect disks, 2. Normalize "unlistable" programs, 3. Use bit-copy programs to make back-ups of the "uncopiables," 4. Write your own adventure-arcade games, 5. Market your software, and 6. Learn all about DOS.

"HARDCORE Computing warns pirates about the latest technology that companies are using against them." **TIME**, Feb. 8, 1982

"When some Apple enthusiasts heard about the boycott (of bit-copy ads), they concluded that it was nothing but censorship and another example of the magazines ignoring the average Apple user to placate their advertisers. So they started their own publication, HARDCORE Computing " **ESQUIRE**, Jan. 1982

HARDCORE COMPUTING
Dept. AL-1
P.O. Box 44549
Tacoma, WA 98444

Subscriptions:
$20.00 U.S.A. $28.50 Canada
$32.50 Mexico $42.00 Others

NAME _____

ADDRESS _____

CITY _____ STATE _____ ZIP _____

Search for Page-Zero References.....Bob Sander-Cederlof

Many times I have wanted a utility which would list out all references to page-zero locations withing a program. For example, when I am trying to avoid conflicts with DOS or Applesoft, I need to know which ones they use and where.

The following little program hooks into the Apple Monitor through the control-Y user command. You type in the address range you want to search through, control-Y, and a carriage return. The Apple will disassemble only those instructions within the address range which reference page-zero locations.

Lines 1220-1280 set up the control-Y vector. When the monitor detects a control-Y command, it branches to $3F8. The JMP instruction there in turn branches to CTRL.Y at line 1320.

Line 1330 loads the first address of the range into PCL and PCH. If you did not type any range before the control-Y, the previous value will be used.

Lines 1340-1540 decide whether the instruction starting at the address in PCL,PCH references page-zero or not. All instructions which reference page-zero have opcodes of the form x1, x4, x5, or x6. All of the x1, x5, and x6 possiblities are valid; only 24, 84-C4, and E4 in the x4 column are valid.

Lines 1580 and 1590 call on a piece of the monitor L-command to disassemble the one instruction. This also updates PCL,PCH to point to the next opcode byte.

Lines 1600-1700 allow you to stop/start the listing by typing any key, to single-step the listing by pressing any two keys simultaneously, and to abort by typing RETURN.

Lines 1740-1780 are executed if the instruction does not reference page-zero. The call on pieces of the L-command to figure out the number of bytes in the instruction and update PCL,PCH accordingly.

Lines 1820-1870 check to see if the range you specified has been covered yet. If not, keep searching; if so, stop.

This kind of program should be in your tool-kit when you are debugging. Just don't lose it under all those other tools!

```
                1010 *-------------------------------
                1020 *      SEARCH FOR PAGE ZERO REFERENCES
                1030 *-------------------------------
003C-           1040 MON.A1L    .EQ $3C
003D-           1050 MON.A1H    .EQ $3D
003E-           1060 MON.A2L    .EQ $3E
003F-           1070 MON.A2H    .EQ $3F
003A-           1080 MON.PCL    .EQ $3A
003B-           1090 MON.PCH    .EQ $3B
                1100 *-------------------------------
C000-           1110 KEYBOARD   .EQ $C000
C010-           1120 STROBE     .EQ $C010
```

```
                        1130 *--------------------------------
FE63-                   1140 MON.LIST2  .EQ $FE63
F88C-                   1150 MON.INSDS  .EQ $F88C
FE75-                   1160 MON.A1PC   .EQ $FE75
F953-                   1170 MON.PCADJ  .EQ $F953
FCBA-                   1180 MON.NXTA1  .EQ $FCBA
                        1190 *--------------------------------
                        1200 *       SET UP CONTROL-Y VECTOR
                        1210 *--------------------------------
0800- A9 4C             1220 SETUPY LDA #$4C      'JMP' OPCODE
0802- 8D F8 03          1230        STA $3F8
0805- A9 10             1240        LDA #CTRL.Y
0807- 8D F9 03          1250        STA $3F9
080A- A9 08             1260        LDA /CTRL.Y
080C- 8D FA 03          1270        STA $3FA
080F- 60                1280        RTS
                        1290 *--------------------------------
                        1300 *       CONTROL-Y COMES HERE
                        1310 *--------------------------------
                        1320 CTRL.Y
0810- 20 75 FE          1330        JSR MON.A1PC IF ADDRESS SPECIFIED, PUT IN PC
0813- A0 00             1340 .1     LDY #0
0815- B1 3A             1350        LDA (MON.PCL),Y
0817- 29 0F             1360        AND #$0F
0819- C9 01             1370        CMP #1
081B- F0 20             1380        BEQ .3
081D- C9 04             1390        CMP #4
081F- 90 3A             1400        BCC .6
0821- D0 16             1410        BNE .2
0823- B1 3A             1420        LDA (MON.PCL),Y
0825- 29 F0             1430        AND #$F0
0827- C9 20             1440        CMP #$20      BIT Z
0829- F0 12             1450        BEQ .3
082B- C9 80             1460        CMP #$80
082D- 90 2C             1470        BCC .6        NO
082F- C9 D0             1480        CMP #$D0
0831- F0 28             1490        BEQ .6        NO
0833- C9 F0             1500        CMP #$F0
0835- F0 24             1510        BEQ .6        NO
0837- D0 04             1520        BNE .3        YES
0839- C9 07             1530 .2     CMP #7
083B- B0 1E             1540        BCS .6
                        1550 *--------------------------------
                        1560 *       INSTRUCTION REFERENCES PAGE-ZERO
                        1570 *--------------------------------
083D- A9 01             1580 .3     LDA #1        DISASSEMBLE THIS ONE INSTRUCTION
083F- 20 63 FE          1590        JSR MON.LIST2   DISASSEMBLE
0842- AD 00 C0          1600        LDA KEYBOARD    SEE IF KEYPRESS
0845- 10 20             1610        BPL .7        NO
0847- 8D 10 C0          1620        STA STROBE    YES, CLEAR IT
084A- C9 8D             1630        CMP #$8D
084C- F0 0C             1640        BEQ .5
084E- AD 00 C0          1650 .4     LDA KEYBOARD
0851- 10 FB             1660        BPL .4
0853- 8D 10 C0          1670        STA STROBE
0856- C9 8D             1680        CMP #$8D
0858- D0 0D             1690        BNE .7
085A- 60                1700 .5     RTS
                        1710 *--------------------------------
                        1720 *       DOES NOT REFERENCE PAGE-ZERO
                        1730 *--------------------------------
085B- A2 00             1740 .6     LDX #0
085D- 20 8C F8          1750        JSR MON.INSDS     GET LENGTH OF INSTRUCTION
0860- 20 53 F9          1760        JSR MON.PCADJ
0863- 85 3A             1770        STA MON.PCL
0865- 84 3B             1780        STY MON.PCH
                        1790 *--------------------------------
                        1800 *       TEST IF FINISHED
                        1810 *--------------------------------
0867- A5 3A             1820 .7     LDA MON.PCL
0869- C5 3E             1830        CMP MON.A2L
086B- A5 3B             1840        LDA MON.PCH
086D- E5 3F             1850        SBC MON.A2H
086F- 90 A2             1860        BCC .1
0871- 60                1870        RTS
```

SUPER PHONE

For Applesoft and D. C. HAYES MICROMODEM II users only

SUPER PHONE IS IN CT, NJ, NY, GA, MN, TX, WI.

UPLOADS, DOWNLOADS, CAPTURES.

KEEPS DIRECTORY

DIALS AUTOMATICALLY

FOR WEEKEND DELIVERY SEND $25

FOR EVENING DELIVERY SEND $27

FOR DAYTIME DELIVERY SEND $30

SEND NAME, ADDRESS, PHONE NUMBER AND TIME TO CALL

JAMES O. CHURCH

20 PLACID ST.

TRUMBULL, CT.

06611

DELIVERY BY PHONE ONLY.

MADE MONEY BACK OFFER WITH FIRST DELIVERIES NO TAKERS YET

Automatic CATALOG for S-C Macro Assembler.....Bill Morgan

Being a thoroughly lazy (and fumblefingered) typist, I have been itching for an automatic CATALOG command to go with the automatic LOAD in the S-C Macro Assembler. Well I finally have it; now loading a file is just esc-C, esc-I...IL. I chose esc-C for CATALOG because I never use the esc-ABCD cursor moves. If you do like those, esc-G and -H are available; right now they are like NOP's.

The Macro Assembler takes the character following an escape (@, A, B,..., L, M) and makes it an index into a jump table located from $1467-1482. Esc-C is at $146D in the table, esc-G is $1475, and esc-H is $1477.

The patch is only $28 bytes long, short enough to easily fit in page 3, but I decided to go ahead and create a spare page for patches by moving the symbol table up one page. This technique is mentioned on page 5-3 of the Macro Assembler manual.

To install the patch, first move the symbol table base up by changing location $101D from $32 to $33. Now insert the address of the patch into the jump table by changing locations $146D-6E from $65 FC to $FF 31 (or your location-1). Type "BLOAD PATCH", then "BSAVE ASM MACRO.MOD,A$1000,L$22FF", and there you have it.

```
                        1000 *--------------------------------
                        1010         .OR $3200
                        1020         .TF PATCH
                        1030 *--------------------------------
0024-                   1040 CH      .EQ $24
0028-                   1050 BASL    .EQ $28
0040-                   1060 XSAVE   .EQ $40
0200-                   1070 WBUF    .EQ $200
                        1080 *--------------------------------
                        1090 ESCAPE.C
3200- E0 00             1100         CPX #0         BEGINNING OF LINE?
3202- D0 1C             1110         BNE .2         NO, RETURN
3204- A0 01             1120         LDY #1
3206- B9 20 32          1130 .1      LDA MSG-1,Y    GET CHARACTER
3209- 91 28             1140         STA (BASL),Y   PUT ON SCREEN
320B- 9D 00 02          1150         STA WBUF,X     PUT IN BUFFER
320E- C8                1160         INY
320F- E8                1170         INX
3210- C0 08             1180         CPY #8         DONE?
3212- D0 F2             1190         BNE .1         NO
3214- 84 24             1200         STY CH
3216- 86 40             1210         STX XSAVE      TELL THE ASSEMBLER
3218- BA                1220         TSX            THAT THIS WAS AN
3219- A9 CC             1230         LDA #$CC       ESCAPE-L, SO IT WILL
321B- 9D 03 01          1240         STA $103,X     GO AHEAD AND EXECUTE
321E- A6 40             1250         LDX XSAVE      THE COMMAND
3220- 60                1260 .2      RTS
                        1270 *--------------------------------
3221- C3 C1 D4
3224- C1 CC CF
3227- C7                1280 MSG     .AS -/CATALOG/
```

Examiner.....Bill Morgan

Here is the program I like to use to examine memory; it displays an entire page on the screen in both hex and ASCII formats. This makes the screen kind of crowded, but I particularly wanted a full page at a time. A program like this is useful for inspecting the results of last month's TRACK READ program, studying the internal format of an Applesoft program, or just exploring inside your Apple.

Examiner uses the left and right arrow keys to decrement or increment the page being displayed. You can also type "P" to allow entry of a page number in hex. Notice that the number entered is rolled into the page number from the right. Escape exits the program.

Lines 1180-1260 set things up to start with page zero.

Lines 1280-1390 display the index, then twelve bytes in hex format.

Lines 1410-1460 reset the indices to display the same twelve bytes in ASCII.

Lines 1480-1630 do the ASCII display, changing any inverse or flashing values to normal and substituting periods for control characters.

Lines 1700-1870 process the commands to change the page being displayed.

Lines 1890-2160 accept characters "0" through "F" and convert them into hex values, rolling the values into the page number to be displayed.

Lines 2180-2260 display the header "page=".

This is threatening to turn into a monthly column; what do you readers think of that idea? Are these routines too trivial? Too complicated? Do you have any questions about them? About anything fairly basic? Drop me a line here at AAL and let me know what you think. I'll look forward to hearing from you.

```
                1000 *--------------------------------
                1010         .OR $300
                1020         .TF EXAMINER
                1030 *--------------------------------
0000-           1040 POINT   .EQ $00,01
0001-           1050 PAGE    .EQ $01
0024-           1060 CH      .EQ $24
                1070 *
C000-           1080 KEYBOARD .EQ $C000
C010-           1090 STROBE   .EQ $C010
                1100 *
F94A-           1110 PRBL2   .EQ $F94A
FC58-           1120 HOME    .EQ $FC58
FD0C-           1130 RDKEY   .EQ $FD0C
FD8E-           1140 CROUT   .EQ $FD8E
FDDA-           1150 PRBYTE  .EQ $FDDA
FDED-           1160 COUT    .EQ $FDED
```

```
                    1170 *-------------------------------
0300- A9 00         1180 START   LDA #$00
0302- 85 00         1190         STA POINT     START WITH
0304- 85 01         1200         STA PAGE      PAGE ZERO
                    1210 *-------------------------------
                    1220 DISPLAY.NEW.PAGE
0306- 20 58 FC      1230         JSR HOME
0309- 20 B3 03      1240         JSR PRINT.HEADER
030C- 20 8E FD      1250         JSR CROUT
030F- A0 00         1260         LDY #$00
                    1270 *
                    1280 NEW.LINE
0311- A2 0C         1290         LDX #$0C      TWELVE BYTES AT A TIME
0313- 98            1300         TYA
0314- 20 DA FD      1310         JSR PRBYTE    PRINT INDEX
0317- A9 A0         1320         LDA #$A0
0319- 20 ED FD      1330         JSR COUT      SPACE
031C- B1 00         1340 .1      LDA (POINT),Y
031E- 20 DA FD      1350         JSR PRBYTE    PRINT HEX
0321- C8            1360         INY
0322- F0 2D         1370         BEQ FILLIN    PAGE DONE?
0324- CA            1380         DEX
0325- D0 F5         1390         BNE .1        TWELVE YET?
                    1400 *
0327- 98            1410 ADJUST  TYA
0328- E9 0C         1420         SBC #$0C      RESET Y
032A- A8            1430         TAY
032B- A9 A0         1440         LDA #$A0
032D- 20 ED FD      1450         JSR COUT      SPACE
0330- A2 0C         1460         LDX #$0C      TWELVE AGAIN
                    1470 *
0332- B1 00         1480 ASCII   LDA (POINT),Y
0334- C9 40         1490         CMP #$40      INVERSE?
0336- B0 02         1500         BCS .1        NO
0338- 09 C0         1510         ORA #$C0      NORMALIZE
033A- C9 80         1520 .1      CMP #$80      FLASHING?
033C- B0 02         1530         BCS .2        NO
033E- 09 80         1540         ORA #$80      NORMALIZE
0340- C9 A0         1550 .2      CMP #$A0      CONTROL?
0342- B0 02         1560         BCS .3        NO
0344- A9 AE         1570         LDA #$AE      PUT PERIOD
0346- 20 ED FD      1580 .3      JSR COUT      SEND IT
0349- C8            1590         INY
034A- F0 0E         1600         BEQ GET.COMMAND   PAGE DONE?
034C- CA            1610         DEX
034D- D0 E3         1620         BNE ASCII     LINE DONE?
034F- F0 C0         1630         BEQ NEW.LINE
                    1640 *
0351- A2 10         1650 FILLIN  LDX #$10      FILL LAST PARTIAL
0353- 20 4A F9      1660         JSR PRBL2     LINE WITH SPACES
0356- A0 08         1670         LDY #$08      ADJUST Y
0358- D0 CD         1680         BNE ADJUST
                    1690 *-------------------------------
                    1700 GET.COMMAND
035A- 20 8E FD      1710         JSR CROUT
035D- 20 0C FD      1720 .1      JSR RDKEY
0360- C9 9B         1730         CMP #$9B      ESCAPE?
0362- F0 0E         1740         BEQ .2
0364- C9 95         1750         CMP #$95      RIGHT ARROW?
0366- F0 0B         1760         BEQ .3
0368- C9 88         1770         CMP #$88      LEFT ARROW?
036A- F0 0C         1780         BEQ .4
036C- C9 D0         1790         CMP #$D0      "P"?
036E- F0 0D         1800         BEQ GET.PAGE.NUMBER
0370- D0 EB         1810         BNE .1        NONE OF THE ABOVE
0372- 60            1820 .2      RTS
                    1830 *
0373- E6 01         1840 .3      INC PAGE
0375- 4C 06 03      1850         JMP DISPLAY.NEW.PAGE
0378- C6 01         1860 .4      DEC PAGE
037A- 4C 06 03      1870         JMP DISPLAY.NEW.PAGE
                    1880 *-------------------------------
                    1890 GET.PAGE.NUMBER
037D- 20 B3 03      1900         JSR PRINT.HEADER
0380- C6 24         1910 .1      DEC CH        SO PRBYTE WILL ALWAYS
0382- C6 24         1920         DEC CH        DISPLAY IN SAME PLACE
```

```
0384- AD 00 C0 1930 .2     LDA KEYBOARD
0387- 10 FB    1940        BPL .2
0389- 8D 10 C0 1950        STA STROBE
038C- C9 8D    1960        CMP #$8D      RETURN?
038E- F0 20    1970        BEQ .5        YES, EXIT
0390- 49 B0    1980        EOR #$B0
0392- C9 0A    1990        CMP #$A       0-9?
0394- 90 06    2000        BCC .3        YES
0396- 69 88    2010        ADC #$88
0398- C9 FA    2020        CMP #$FA      A-F?
039A- 90 E8    2030        BCC .2        NO
               2040 *
039C- A0 03    2050 .3     LDY #$3       LOOP 4 TIMES
039E- 0A       2060        ASL           THROW AWAY HIGH NYBBLE
039F- 0A       2070        ASL
03A0- 0A       2080        ASL
03A1- 0A       2090        ASL
03A2- 0A       2100 .4     ASL           SHIFT INTO
03A3- 26 01    2110        ROL PAGE      PAGE NUMBER
03A5- 88       2120        DEY
03A6- 10 FA    2130        BPL .4
03A8- A5 01    2140        LDA PAGE
03AA- 20 DA FD 2150        JSR PRBYTE    DISPLAY PAGE NUMBER
03AD- 4C 80 03 2160        JMP .1        GET NEXT KEYPRESS
03B0- 4C 06 03 2170 .5     JMP DISPLAY.NEW.PAGE
               2180 *-------------------------------
               2190 PRINT.HEADER
03B3- A0 00    2200        LDY #$00
03B5- B9 C5 03 2210 .1     LDA QPAGE,Y
03B8- 20 ED FD 2220        JSR COUT
03BB- C8       2230        INY
03BC- C0 05    2240        CPY #$05
03BE- D0 F5    2250        BNE .1
03C0- A5 01    2260        LDA PAGE
03C2- 4C DA FD 2270        JMP PRBYTE
               2280 *
03C5- D0 C1 C7
03C8- C5 BD    2290 QPAGE  .AS -/PAGE=/
```

Apple Assembly Line is published monthly by S-C SOFTWARE CORPORATION, P. O. Box 280300, Dallas, Texas 75228. Phone (214) 324-2050. Subscription rate is $15 per year in the USA, sent Bulk Mail; $18 per year sent First Class Mail in USA, Canada, and Mexico; $28 per year sent Air Mail to other countries. Back issues are available for $1.50 each (other countries add $1 per back issue for postage). All material herein is copyrighted by S-C SOFTWARE, all rights reserved. (Apple is a registered trademark of Apple Computer, Inc.)